Research Article

Venkatesh SS and Deepak Mishra*

Variable Search Space Converging Genetic Algorithm for Solving System of Non-linear Equations

https://doi.org/10.1515/jisys-2019-0233
Received Oct 10, 2020; accepted Feb 10, 2020

Abstract: This paper introduce a new variant of the Genetic Algorithm which is developed to handle multivariable, multi-objective and very high search space optimization problems like the solving system of non-linear equations. It is an integer coded Genetic Algorithm with conventional cross over and mutation but with Inverse algorithm is varying its search space by varying its digit length on every cycle and it does a fine search followed by a coarse search. And its solution to the optimization problem will converge to precise value over the cycles. Every equation of the system is considered as a single minimization objective function. Multiple objectives are converted to a single fitness function by summing their absolute values. Some difficult test functions for optimization and applications are used to evaluate this algorithm. The results prove that this algorithm is capable to produce promising and precise results.

Keywords: Genetic Algorithm, Roulette Wheel selection, Cross over, Mutation, Multi Objective optimization, Multi variable optimization„ Genetic Algorithm, Multi Objective optimization, Multi variable optimization

PACS: computer science, Artificial intelligence

1 Introduction

Genetic algorithm (GA) can be chosen as a meta-heuristic search algorithm when a best possible solution/s of non-linear and discontinuous functions is to be found. GA does not require complex mathematics to execute and it can find the global optimum even when many local optima available. Genetic algorithm is probabilistic optimization method, that mimics the process of natural selection and it is a kind of evolutionary algorithms (EA). GA generates the solutions using operators such as Selection, Cross Over and Mutation which are inspired by natural evolution. This evolution is governed by a simple law which Charles Darwin named as –“Survival of the Fittest”.

In the computer science field of artificial intelligence, a genetic algorithm (GA) is a search heuristic [8]. This heuristic (also sometimes called a meta-heuristic) is routinely used to generate useful solutions to optimization and search problems. Genetic algorithm is probabilistic optimization method, which generate solutions to optimization problems using techniques inspired by natural evolution, such as inheritance, mutation, selection, and crossover. The term genetic algorithm, almost universally abbreviated nowadays to GA, was first used by John Holland [6], whose book Adaptation in Natural and Artificial Systems of 1975 was instrumental in creating what is now a flourishing field of research and application that goes much wider. John Holland’s original GA is called as Simple Genetic Algorithm (SGA) which has its limitations to handle multi variable and multi objective functions. Subsequently in [7] JR Koza had extended his work and proposed Genetic Programming, the Programming of Computers by Means of Natural Selection.

***Corresponding Author: Deepak Mishra:** Dept. of Avionics, IIST Trivandrum, Kerala, India; Email: deepak.mishra@iist.ac.in
Venkatesh SS: IIST Trivandrum, Kerala, India

Later, Akira Oyama *et al.* [1] have developed a Real-coded Adaptive Range Gas (ARGAs) to find a solution to an aerodynamic airfoil shape optimization problem. The results show that the real-coded ARGAs find better solutions than the conventional real-coded GAs. Subsequently, F.Herrera and M.Lozano [2] have developed a Two-loop Real-coded GA with Adaptive Control of Mutation Step Sizes (TRAMSS). TRAMSS adjusts the step size of a mutation operator applied during the inner loop, for producing efficient local turning, thus avoids premature convergence. It also controls the step size of a mutation operator used by a restart operator performed in the outer loop, for re-initializing the population in order to ensure that different promising search zeroes are focused by the inner loop throughout the run. Solving difficult nonlinear equation using evolutionary technique is an important research area [11]. Many researcher have attempted this by using neural network and genetic algorithms. The work carried by Crina Grosan and Ajith Abraham [3], presents an evolutionary approach for Solving Nonlinear Equations Systems. They propose an evolutionary technique for solving systems of complex nonlinear equations by simply viewing them as a multi-objective optimization problem. Their approach has given good results for benchmark problems compared to earlier results. Later Satoshi Tomioka *et al.* [4] also proposed an adaptive domain method (ADM) using real-coded GAs to solve non-linear problems. They have demonstrated the effectiveness of the new method by citing an example problem. Recently parameter optimization in GA found to be very important. The book [5] describes in detail some aspects of GA parameter optimization for recent applications.

Solving system of equations is very old and an important research problem and many researchers have given important consideration [14–17]. In 2002 the paper [18] deploys a gradient descent approach to solve nonlinear system of equations. Authors in [24, 25] have used a Hopfield type neural network and energy function approach for finding roots of characteristic equation. The computational cost in GA based nonlinear equations solver is an important concern there are many variants to GA have been proposed in the past. An excellent survey on various methods can be found at [26]. Authors in this paper did an excellent job on summarizing the various works related to this area. The article [20] presents the estimation of roots of nonlinear equations using Genetic Algorithm varying the population size, crossover rate, degree of mutation, and coefficient size. In this paper [21], the author first converted single and simple set of nonlinear system of equations into unconstrained optimization problem, and complex set of systems into constrained optimization problem. Afterward, Genetic Algorithm tool is applied to solve the system. In a recent work [22] the author developed a new approach in which optimum solution of nonlinear system of equations is obtained by a method based on variants of Genetic Algorithm using evolutionary computational technique. The paper [23] describes a novel application of Genetic Algorithm for approximating solution of optimum problems by introducing pairs of harmonious and symmetric individuals. Although many methods have been proposed this paper tries to propose the application of variable search space in finding the solutions of system of equations. While finding solution to system of non-linear equations, the search algorithm has to handle multiple variables, multiple objectives and a very high search space. Hence a new Genetic Algorithm named as 'Variable Search Space Converging Genetic Algorithm (VSSCGA)' is developed, which efficiently handles multiple variables, multiple objectives and high search space. The VSSCGA has been validated with benchmark problems and got very good results compared to previous results.

2 Search Space

Let the VSSCGA has integer coded representation and each variable (x) is a chromosome and its each gene has integer numbers from 0 to 9. Here the chromosome's length or number of genes in a chromosome is called as Digit Length (nD), which is similar to Bit Length in binary representation. Let $nD = 1$. Hence the only possible chromosome representations are $0, 1, 2, 3, .., 9$. Here it can be say that the VSSCGA's Representation Range (VRR) is 0 to 9. And if $nD = 2$, the possible chromosome representations are $00, 01, 02, \ldots, 09, 10, 11, \ldots, 99$. Here the VRR is 00 to 99. Thus if =3, then the VRR is $[000, 999]$, and so on.

It is assumed that the integer coding representation of chromosome is following the 'base 10' number system. If the Search Space $(\Omega)'s$ Range (SSR) is assumed as $[0, 1)$, and $nD = 1$, then each chromosome representation's Value within Range (V_R) in SSR would be $0 \to 0.0$, $1 \to 0.1$, $2 \to 0.2$,..., $9 \to 00.9$. And if SSR is [0, 1), and $nD = 2$, then each chromosome representation's Range value in SSR would be $00 \to 0.00$, $01 \to 0.01$, $02 \to 0.02$,..., $09 \to 0.09$, $10 \to 0.10$, $11 \to 0.11$,...,$99 \to 0.99$. Here, if a chromosome is having representation as , it is actually a string of '36'. This value can be named as Value of String ($'36'$). For example, if $nD = 5$, SSR=[0, 1) and if one chromosome's representation is 36159, its String Value ='36159' and Range Value within SSR =0.36159. In general, if $nD = n$ and a variable string is '$d_n d_{n-1}, ..., d_2, d_1'$, the String Value $V_s = d_n \times 10^{n-1} + d_{n-1} \times 10^{n-2}, \ldots, +d_2 \times 10^1 + d_1 \times 10^0$. For example, if a variable is having string as '5 2 6 2 4 6', its decimal value is $5 \times 10^5 + 2 \times 10^4 + 6 \times 10^3 + 2 \times 10^2 + 4 \times 10^1 + 6 \times 10^0 = 526246$. Then the Value within Range can be computed as

$$V_R = V_S \frac{SSR_{max} - SSR_{min}}{10^{nD}} + SSR_{min}$$

For example if $SSR_{min} = 0$, $SSR_{max} = 10$, $nD = 6$ and $V_S = 526246$, then $V_R = 5.26246$.

Consider n objective functions such a way that, $f_i(x_j) : \Omega \to R$, where $x_j \in \Omega$, and $\Omega \in R^m$, $i =, 1, 2, \ldots, n$ and $j = 1, 2, \ldots, m$

To visualize the search space Ω , first let $i = 1$ and $j = 1$, i.e. Single Objective and Single Variable function. Then if $nD = 1$, the variable x_i can pick a value from 10 number of solutions. Thus if $nD = n$, the variable x_i can pick a value from 10^n number of solutions. It means the Search Space has 10^n number of possible solutions. These explanations show that in integer coding, by varying the Digit Length, the Search Space Size (SSS) can be varied.

As another example let us assume that the problem has 8 variables and all the 8 variables of the solution are rounded to 3 decimal places. Then the GA has to find a most fitted solution among $10^{3\times8} = 10^{24}$ solutions or combinations. If the solution set is rounded to 6 decimal places, then the GA has to find a most fitted solution among $10^{6\times8} = 10^{48}$ solutions. Hence the required GA should have very high searching capability. And since it needs to handle multi variables, it should have better capability to exit/jump from the local minima than the mutation operator provides. With these requirements a new GA called 'Variable Search Space Converging Genetic Algorithm' is developed.

3 Variable Search Space Converging Genetic Algorithm

The main concept of this VSSCGA is, it varies its digit length over its cycles and thus it varies its search space. In its first cycle (i.e. first digit cycle), its population is represented with a single (which can be varied) digit number (integer) ranging from 0 to 9, for each variable of the population individual. Its second cycle works with two digit numbers ranging from 0 to 99 and the third cycle works with three digit numbers ranging from 0 to 999 and so on. The initial/lower cycles provide a platform for coarse search and higher digit cycles account for fine search. The term converging here meant is that the process of growth chromosome will not be of fixed length. It will gradually attain the maximum allowed length. It may also happen that the growth of chromosome digit length will be stopped when the algorithm attains the desired solutions.

3.1 Algorithms and Features

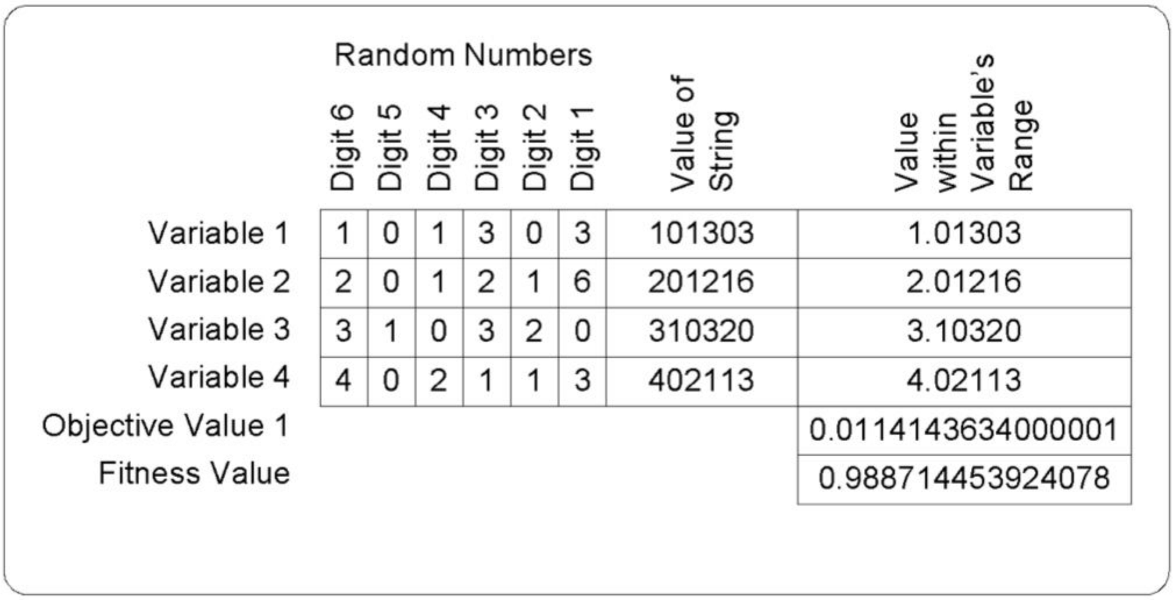
The VSSCGA has many advantages over conventional binary coded GA, which can be summarized as follows [3]:

- The solutions can be represented more precisely and thus the computation complexity is amended and the computation efficiency is improved.
- Since during initial cycles the SSS is low, the execution time is faster. It takes lesser time to arrive the solution to the expected precision and fitness.
- It avoids the premature convergence always by its selection procedure.

- As the design variables are coded by floating numbers in classical optimization algorithms, the VSSCGA is more convenient for combination with classical optimization algorithms.

The algorithmic details of the proposed VSSCGA are presented in the Appendix: A. Here, we describe the various important variables are the necessary steps involved to reproduce the proposed algorithm. The salient features of VSSCGA:

1. It works on three types of loops, one over another.
2. The inner loop is Iteration Loop, which is based on SGA.
3. The middle loop is Round. Multiple iterations form a Round. Every new Round starts with a random new population. Multiple Rounds are designed to introduce multiple random new populations over the complete run.
4. The outer loop is Digit Cycle. Multiple rounds form a Digit Cycle. Multiple Digit Cycles of Rounds is designed to take care of coarse search and fine search.
5. Inverse Ranked Roulette wheel selection is designed to take care of premature convergence and thus to improve the search capability.
6. Elitism between Iterations and Migration between Rounds are designed to avoid the best solution getting destroyed.
7. Stopping criterion based on Maximum Iteration, Consistency of best solution over the iterations and Saturation of Fitness over the population of a generation are designed to stop the algorithm.

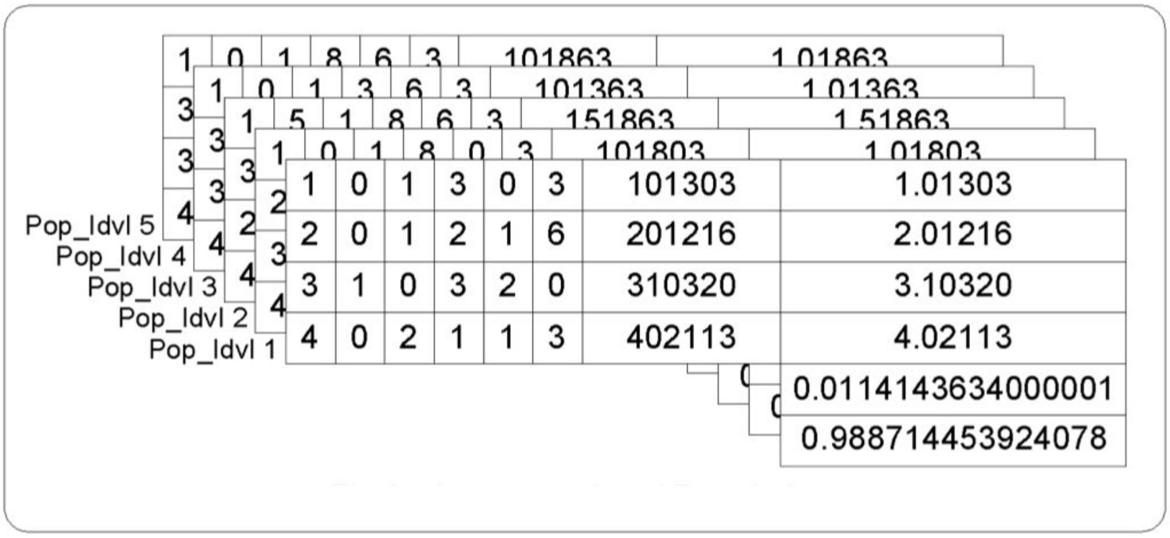


	Random Numbers							
	Digit 6	Digit 5	Digit 4	Digit 3	Digit 2	Digit 1	Value of String	Value within Variable's Range
Variable 1	1	0	1	3	0	3	101303	1.01303
Variable 2	2	0	1	2	1	6	201216	2.01216
Variable 3	3	1	0	3	2	0	310320	3.10320
Variable 4	4	0	2	1	1	3	402113	4.02113
Objective Value 1								0.0114143634000001
Fitness Value								0.988714453924078

Figure 1: Structure of one Population Individual

3.2 Structure of Population Individual

In order to GA operators work well with the Population Matrix, and to the objective and fitness values also to get integrated with the Population Matrix, it's structure is designed in Matlab as shown Figure 1. An example of the structure of a Population Individual for $nV = 4, nD = 6, nP = 5, nO = 1$ is shown in Figure 1 and its corresponding total population structure is shown in Figure 2. It is a three dimensional matrix having its first dimension (row) as nV , second dimension (column) as $nD + 2$ and third dimension (page) as nP. The value within variable's range V_R is computed with an assumption of SSR=[0, 10). The advantage of this integrated


Pop_Idvl 5
Pop_Idvl 4
Pop_Idvl 3
Pop_Idvl 2
Pop_Idvl 1

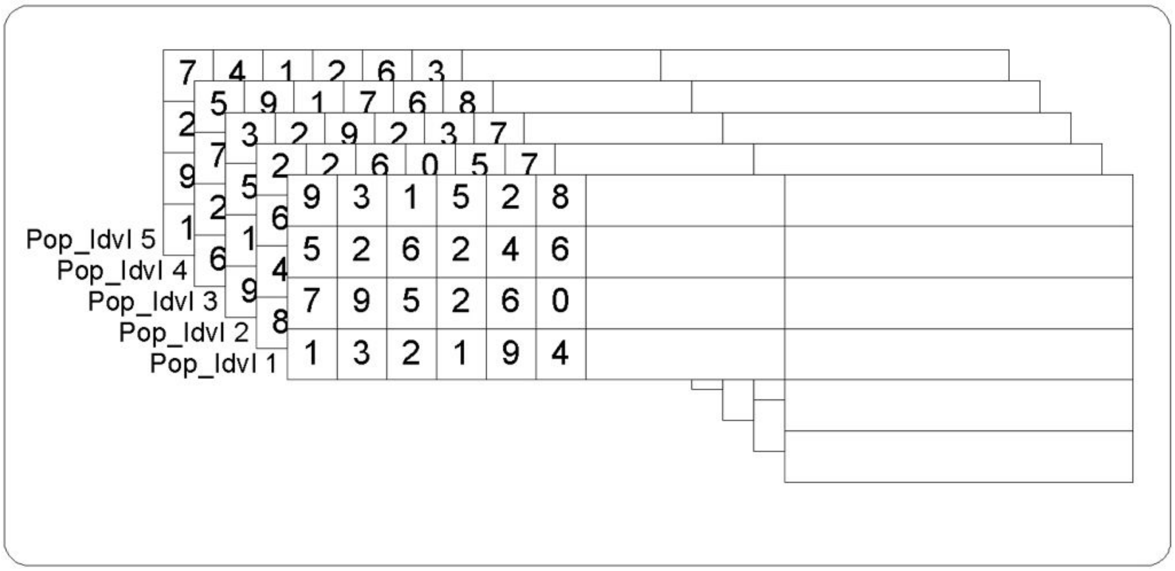
1	0	1	3	0	3	101303	1.01303
2	0	1	2	1	6	201216	2.01216
3	1	0	3	2	0	310320	3.10320
4	0	2	1	1	3	402113	4.02113
							0.0114143634000001
							0.988714453924078

Figure 2: Structure of total Population

structure is when the population individuals are sorted out based on their fitness it needs to operate only one matrix. Similarly when the best population individual having best fitness is getting selected and when the elitism and migration operations are carried out, it is easy to handle/select the population individual matrix which has all its details including its values in the range, objective values and fitness value. On the other hand, if the population and its properties are maintained in separate matrices, handling them in the code is difficult and may give wrong results.

3.3 Initial Population Generation

At the beginning of every round a new population is initialized with random numbers (single digit integers) with dimension $nV \times nD \times nP$. An example of a new random new population is shown in Figure 3.


Pop_Idvl 5
Pop_Idvl 4
Pop_Idvl 3
Pop_Idvl 2
Pop_Idvl 1

9	3	1	5	2	8		
5	2	6	2	4	6		
7	9	5	2	6	0		
1	3	2	1	9	4		

Figure 3: Random Population Initialization

3.4 Value Computation

After new population generation, the values V_S and V_R of each variable of the population individual is computed as explained in section ‘Search Space’.

3.5 Fitness Computation

The Value in range (V_R) of all variables of an individual is fed to Fitness function to evaluate the objective function value. Here it is assumed that all objective function values are of positive values or cost/distance value in nature or it is made as positive by adding ‘absolute’ function or by introducing suitable offset or by some other alternative methods possible.

If the objective of the optimization problem is to maximize, and if there is only one objective function, the Value of the Objective function is directly taken as the Fitness function value . And if more than one objective function is there, the sum of each objective function value is taken as i.e. $V_F = V_{o1} + V_{o2} + V_{o3}$.

If the objective of the optimization problem is to minimize, and if there is only one objective function, the value of the Fitness function V_F is calculated as $V_F = \frac{1}{1+V_o}$. And if more than one objective function is there, the Fitness function value V_F is calculated as $V_F = \frac{1}{1+(V_{o1}+V_{o2}+V_{o3})}$. Fitness function value is very important in GA, as it plays a vital role in selection of population for next generation.

3.6 Selection – Inverse Ranked Roulette Wheel

Selection operator selects individuals in the population for reproduction. The fitter the individuals, the more times it is likely to be selected to reproduce. Selection operator works on the principle of Charles Darwin’s ‘Survival of the Fittest’. Many types of selection are discussed in the literature including Roulette wheel selection, Rank selection, Tournament selection etc. Each method has its own advantages and disadvantages.

Here a combination of Roulette wheel selection with Rank selection is introduced. This algorithm assumes that all $V_F \geq 0$. First the individuals are sorted in descending order depending on their fitness value. In this order, the highest fit individual has the first place and the lowest fit individual has the last place (i.e. nP^{th} place). Then each individual is assigned with a rank of $1/(1+itsplacevalue)$. Hence the name is Inverse Ranked. The highest fit individual or first individual is having rank $\frac{1}{2}$, the j^{th} individual has the rank $1/(1+j)$ and the last individual has the rank $1/(1+nP)$.

Similar to the normal roulette wheel algorithm, in this method also, the probability of fitness of each individual is computed and expected count of each individual is calculated. Then the Ranked Expected Count (r_{ec}) of each individual is computed by multiplying expected count with its rank and the value is rounded up (ceil) to integer values.

The Ranked Expected Count of j^{th} individual

$$r_{ec_j} = ceil\left(\frac{F_{v_j} \times nP}{\sum_{i=1}^{np} F_{v_j}} \times \frac{1}{1+j}\right)$$

The r_{ec} of an individual who is having $V_F = 0$, is assigned as 1. With this expected count the remaining procedure of roulette wheel is carried out. Each individual is reproduced their corresponding r_{ec} times. Then first/top nP individuals are selected for the next generation.

This IRRW selection improves the searching capability of the GA. Its main advantage is no single individual can make the complete population saturated throughout the generations, thus it always avoids premature convergence.

3.7 Cross Over – Binary Imitative

The crossover operator is the most important operator of GA, because it leads to convergence [9]. In crossover, generally two chromosomes, called parents, are combined together to form new chromosomes, called off-spring. Random pairs are selected for cross over from the complete population of IRRW selection. Many real coded cross over types are discussed in the literature. To get advantage of complete randomness, the binary imitative uniform single point cross over is used. Two pseudo random numbers are generated, first being real floating number having range 0 to 1, to check the 'go ahead status' of the cross over and second being integer having range 1 to nD, to decide the cross over point. If the first random number is less than or equal to P_c, the cross over is carried out. The second random integer number gives the place/position of the cross over point. The genes after the cross over points of both parents are interchanged. An example of cross over is shown in Figure 4.

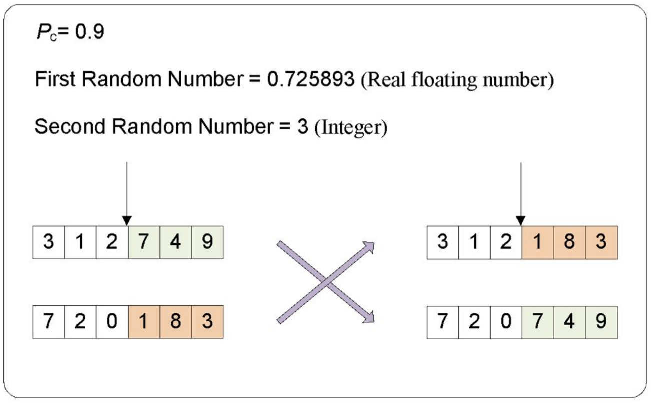

P_c= 0.9
First Random Number = 0.725893 (Real floating number)
Second Random Number = 3 (Integer)
3 1 2 7 4 9
7 2 0 1 8 3
3 1 2 1 8 3
7 2 0 7 4 9

Figure 4: An example of Cross Over

3.8 Mutation – Binary Imitative

In genetic algorithms, mutation realized as a random deformation of alleles with a certain probability. Mutation can occur at each bit position in a string with very small probability (e.g., 1%). Mutation plays a critical role in GA. As discussed earlier, crossover leads the population to converge by making the chromosomes in the population alike. Mutation reintroduces genetic diversity back into the population and assists the search escape from local optima. Many real coded mutation types are discussed in the literature. Similar to cross over to get advantage of complete randomness, the binary imitative uniform single point mutation is used. All individuals from cross over undergo mutation operation. Instead of considering all genes, the whole chromosomes (i.e.variables) are considered for mutation with probability P_m. Thus in each chromosome a single gene may have the chance to get mutated. This will reduce the number of random numbers to be generated and thus improves the execution speed. In order to get the same mutation performance, the mutation probability P_m may be increased. Typically it may have value from $5 to 10\%$. Here in mutation three pseudo random numbers are generated, first being real floating number having range 0 to 1, to check the 'go ahead status' of the mutation and second being integer having range 1 to nD , to decide the mutation point and third also being integer having range 0 to 9, is the value of the digit to be interchanged. If the first random number is less than or equal to P_m, the mutation is carried out. The second random integer number gives the place/position of the mutation point. The gene at the mutation point of the parent is interchanged with the third random integer. An example of mutation is shown in Figure 5.

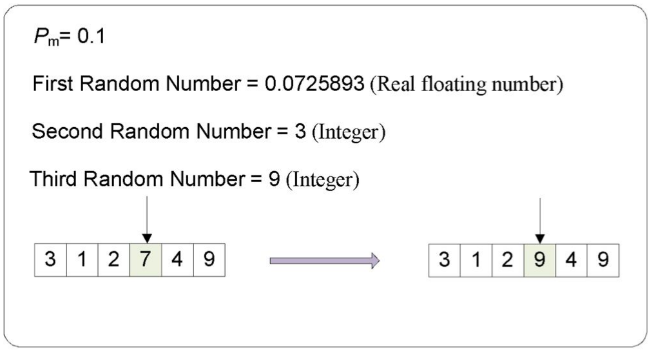

P_m= 0.1
First Random Number = 0.0725893 (Real floating number)
Second Random Number = 3 (Integer)
Third Random Number = 9 (Integer)
3 1 2 7 4 9
3 1 2 9 4 9

Figure 5: An example of Mutation

3.9 Rounds of Iterations

In this VSSCGA, multiple iterations **are called as a round**. Each round starts with a new random population. As such to introduce random new components to the overall algorithm, the algorithm can be run for many rounds. Since it re-initializes the whole population after every round, this prevents premature convergence and gives greater searching capability.

3.10 Varying Digit Length (Precision)

Many rounds form a digit cycle. The new aspect of this VSSCGA is the varying precision (decimal fractions) of variables in every digit cycle of the GA. Each gene of the GA is formed with single digit integer which has range 0 to 9. Multiple genes form the chromosome. Each chromosome represents a variable. In single variable problem, each chromosome/variable forms an individual in a population. In multi variable problem, multiple chromosomes form an individual. The number of genes (or digit length) in a chromosome is decided based on the digit cycle settings. For example, if the digit cycle setting has $St_{dgt} = 1$ and $Sp_{dgt} = 6$, in the first digit cycle, each chromosome has only one gene (i.e. $nD = 1$), second digit cycle has two genes and the last/sixth cycle has 6 genes (i.e. $nD = 6$) in every chromosomes. Throughout a digit cycle, which has multiple rounds and iterations, the number of genes in a chromosome is fixed.

If the range of the GA fitness function variables is $SSR_{min} = 0$ and $SSR_{max} = 10$, and when $nD = 1$, the variables may have values like 0, 1, 2, ……8, 9 (full rounded integer). Here if each individual has number of variables $nV = 8$, the search space size is only 10^8. If $nD = 2$, the variables have values like 0, 0.1, 0.2….1.0, 1.1, 1.2……9.8, 9.9 and here the search space size is $10^{8\times2}$. And similarly for $nD = 3$, the variable values are 0, 0.01, 0.02,…..1.00, 1.01, 1.02, …… 9.98, 9.99 and the search space size is $10^{8\times3}$, and so on.

In the first digit cycle, since the search space is less, GA can find the rounded or approximate value of the solution easily. It is similar to coarse search. In the successive digit cycles, as the search space increase, GA does a fine search or fine tuning of the solution.

3.11 Elitism

De Jong (Jong) [10] suggested a policy to always include the best individual of one generation into next generation in order to prevent losing it due to sampling effects or operator disruption. This strategy, which can be extended to copy the one or more best solutions to the next generation, is denoted as elitism. The idea is to avoid that the observed best-fitted individuals dies out just by selecting it for the next generation without any random experiment. Elitism is widely used for speeding up the convergence of a GA. The negative effect of the elitism i.e. premature convergence is taken care in VSSCGA by introducing IRRW selection.

In order to have more randomness and searching capability, VSSCGA has the provision to start the elitism operator at any iteration (i.e. at first iteration or at higher iterations less than Max_{Itr}). Two settings are specified regarding elitism. First one is number of best individuals ($nElit$) to be preserved for elitism and the second one is iteration count ($Elit_{Itr}$) from which the elitism operation starts. The recommended value for $nElit$ is at least 2 and at the maximum of 0.1×nP. And the recommended value for $Elit_{Itr}$ is $0.5 \times Max_{Itr}$, which gives equal chance for non converging search and converging search. It means that in every round until $0.5 \times (Max_{Itr})^{th}$ iteration, elitism is not working, hence GA works on complete random search in each iteration, with very **little** probability for convergence.

3.12 Migration

Elitism operator takes care of carrying the best individuals to next iterations. An another operator like elitism is required to take care of carrying the best individuals/solution to next rounds and digit cycles. This operator is named as Migration operator. In the first digit cycle migration works after $Round > 1$, and for other digit cycles migration works on all rounds. That is, migration preserves the best individual of the first round of the first digit cycle and carries to the next rounds and next digit cycles. Elitism operator introduces the best individuals in every iteration after $Elit_{Itr}$. But the migration operator injects the best individuals only once at the iteration $Migr_{Itr}$ in a round. If it is ensured that $Migr_{Itr} > Elit_{Itr}$, after the injection of best individuals by migration, the elitism carry them to next iterations. In order to get best converged result, the stopping criterion other than Max_{Itr}, will get activated only after $Migr_{Itr} + 1$ and $Elit_{Itr} + 1$.

The best individuals of migration are carrying forward to next rounds within a digit cycle without any scaling or modification. But when the migration works to next digit cycle, the best individuals of the current digit cycle have to be transformed (i.e. to undergo scaling or suitable transformation) based on the fitness function and variable range, in such a way that the values of variables of the best individual in the current cycle is equal to the values of variables in next cycle.

3.13 Best Solution Selection

The best solution selection operator gives the answer of the GA optimization. Initially the maximum fitness (Max_{Fit}) of the population of the first iteration of the first round of the first digit cycle is stored as $Best_{Fit}$, and its corresponding individual is stored as $Best_{Pop}$. From the next iteration on-wards if the Max_{Fit} of the current iteration is greater than the $Best_{Pop}$, it replaces the $Best_{Fit}$ and corresponding individual replaces the $Best_{Pop}$. The $Best_{Pop}$ gives the solution from one round.

Similarly the $Best_{Fit}$ and $Best_{Pop}$ of the first round are stored as $RBest_{Fit}$ and $RBest_{Pop}$. If the next round's $Best_{Fit}$ is greater than the $RBest_{Fit}$, it replaces the $RBest_{Fit}$ and corresponding individual replaces the $RBest_{Pop}$. The $RBest_{Pop}$ gives the solution of one digit cycle. Thus the $RBest_{Fit}$ and $RBest_{Pop}$ of the first digit cycle are stored as $FBest_{Fit}$ and $FBest_{Pop}$. If the next digit cycle's $RBest_{Fit}$ is greater than the $FBest_{Fit}$, it replaces the $FBest_{Fit}$ and corresponding individual replaces the $FBest_{Pop}$. The $FBest_{Pop}$ gives the solution of the optimization problem or the fitness function.

3.14 Stopping criterion

Three stopping criterion are available within the iteration loop. Two stopping criterion are available within the rounds. And one stopping criterion is there for digit cycle. The stopping criterion of the iteration loop are based on,

1. Maximum number of iterations,
2. Consistency of best fit over the specified number of iterations and
3. Saturation of fitness over the number of individuals in a generation.

The stopping criterion of the round loop are based on,

1. Maximum number of rounds
2. Consistency of best fit over the specified number of rounds.

The stopping criterion of the digit cycle is only based on stop digit.

The iteration loop stops at the Max_{Itr}, irrespective of any other condition of the GA. The iteration loop stops, when the best fit of iterations is consistent over the specified number of iterations ($nCons_{Itr}$). If the specified number of individuals ($nSatu_{Pop}$) in a generation population are having the same fitness value (fitness saturation), then also the iteration loop stops. A minimum fitness (min_{Fit}) threshold is provided and until GA achieves this threshold, the last two stopping criterion will not get activated. To implement this provision, the user should have the knowledge of minimum fitness value which can be achieved as $Best_{Fit}$. As it is discussed in the migration section, the last two stopping criterions (i.e. other than Max_{Itr}) will get activated only after $Migr_{Itr}$ + 1 and $Elit_{Itr}$ + 1 to get best converged result.

The round loop stops at the Max_{Rnd}, irrespective of any other condition of the GA. And the round loop stops, when the best fit of rounds is consistent over the specified number of rounds ($nCons_{Rnd}$). The digit cycle starts at St_{dgt} and stops with Sp_{dgt}.

4 Tests and Results

In order to verify the effectiveness of this VSSCGA, it is tested with two Effati and Nazemi examples, 4 test functions for optimization and two benchmark optimization problems and the results are obtained.

4.1 Effati and Nazemi Examples

4.1.1 Effati and Nazemi Example 1

The Effati and Nazemi example 1 (Effati.S) [11] contains a system of two equations with two variables. The variable's range used in VSSCGA is $0 \le x_1, x_2 \le 1$.

$$f_1(x_1, x_2) = cos(2x_1) - cos(2x_2) - 0.4 = 0$$
$$f_2(x_1, x_2) = 2(x_1 - x_2) + sin(2x_1) - sin(2x_2) - 1.2 = 0$$

The settings of VSSCGA for this example are given in Table 1. The results of every digit cycle are shown in Table 2. The comparison of VSSCGA's results with earlier results (Abraham) is shown in Table 3. In Table 3 we have performed a comparison of the popular numerical methods and GA based methods including the proposed one on Effati and Nazemi Example function. We show here the values of xi and Functions values and moreover, the function values with the proposed method are very accurate. Further, it is interesting to observe in Table 2 that on how the variations in nD affect the accuracy and precision of the solution. It is also seen here that going to a higher digit length is not required in certain cases. Hence, the variable search space idea does help in reducing the complexity implicitly without affecting the solution accuracy.

Table 1: VSSCGA Settings for Effati and Nazemi example 1

nV	nO	St_{dgt}	Sp_{dgt}	nP	Max_{Itr}	Max_{Rnd}	Pc	Pm
2	2	1	10	100	100	4	0.9	0.1

The result achieved in Table 2 is only upto Sp_{dgt} = 10. If the VSSCGA is allowed to do more number of digit cycles, it can give better precise results.

Table 2: Digit Cycle Results for Effati and Nazemi example 1

nD	Solution x_i	Function Value f_i	Fitness Value
1	0.2, 0.5	−0.0192413118652547, −0.147947357500754	0.85675951647404
2	0.17, 0.51	0.0193887142766966, −0.00137907019145156	0.979654741475831
3	0.157, 0.494	0.000744905890763, 0.000061407460612	0.999194336266051
4	0.1571, 0.4941	0.000850110156655, −0.000018752041260	0.999131892068250
5	0.15652, 0.49338	0.000006092467513, 0.000011522010458	0.999982385832294
6	0.15652, 0.493376	−0.000000581465550, −0.000000889181838	0.999998529354774
7	0.1565201, 0.4933764	0.000000024335686, −0.000000038340559	0.999999937323759
8	0.1565201, 0.49337641	0.000000041020483, −0.000000007312525	0.999999951666995
9	0.1565201, 0.493376412	0.000000044357443, −0.000000001106918	0.999999954535641
10	0.1565201, 0.4933764123	0.000000044857987, −0.000000000176077	0.999999954965938

Table 3: Comparison of VSSCGA's results with earlier results for Effati and Nazemi example 1

S.no.	Method	Solution x_i	Functions values f_i
1	Newton	(0.15, 0.49)	(−0.00168, 0.01497)
2	Secant	(0.15, 0.49)	(−0.00168, 0.01497)
3	Broyden	(0.15, 0.49)	(−0.00168, 0.01497)
4	Effati (Effati.S)	(0.1575, 0.4970)	(0.005455, 0.00739)
5	E. A. (Abraham)	(0.15772, 0.49458)	(0.001264, 0.000969)
6	VSSCGA	(0.1565201, 0.4933764123)	(0.000000044857987, −0.000000000176077)

4.1.2 Effati and Nazemi Example 2

The Effati and Nazemi Example 2 (Effati.S) also contains a system of two equations with two variables. The variable's range used in VSSCGA is $0 \le x_1, x_2 \le 10$.

$$f_1(x_1, x_2) = e^{x_1} + x_1 x_2 - 1 = 0$$
$$f_2(x_1, x_2) = sin(x_1 x_2) + x_1 + x_2 - 1 = 0$$

The settings of VSSCGA for this example are given in Table 4. The results of every digit cycle are shown in Table 5 . The comparison of VSSCGA's results with earlier results (Abraham) is shown in Table 6.

Table 4: VSSCGA Settings for Effati and Nazemi example 2

nV	nO	St_{dgt}	Sp_{dgt}	nP	Max_{Itr}	Max_{Rnd}	Pc	Pm
2	2	1	4	20	50	4	0.9	0.1

Table 5: Digit Cycle Results for Effati and Nazemi example 2

nD	Solution x_i	Function Value f_i	Fitness Value
1	0, 1	0, 0	1
2	0, 1	0, 0	1
3	0, 1	0, 0	1
4	0, 1	0, 0	1

Table 6: Comparison of VSSCGA's results with earlier results for Effati and Nazemi example 2

S.no.	Method	Solution x_i	Functions values f_i
1	Effati (Effati.S)	(0.0096, 0.9976)	(0.019223, 0.016776)
5	E. A. (Abraham)	(−0.00138, 1.0027)	(−0.00276,−0.0000637)
6	VSSCGA	(0, 1)	(0, 0)

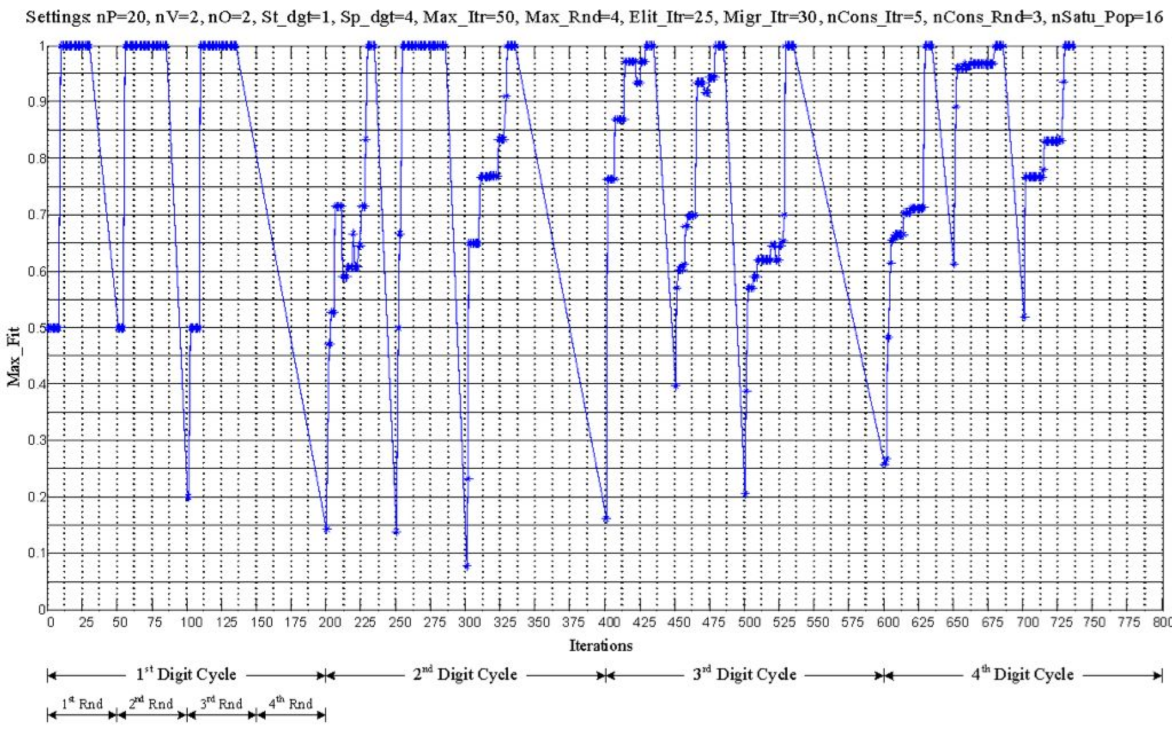


Figure 6: Flow of max_{Fit} over iterations, for Effati and Nazemi Example 2

The flow of Max_{Fit} over iterations, rounds and digit cycles is shown in Figure 6. It can be noted that VSSCGA has found the solution in 10^{th} iteration of the first round and first digit cycle itself. This graph explains many concepts of VSSCGA.

4.2 Test functions for optimization

Since the efficiency of VSSCGA for two variables and two objective test functions is proved, the VSSCGA is going to be tested with higher dimension test functions and to precisely estimate the values of Schaffer 4 function than the values given in the literature [13].

4.2.1 Ackley's function

The function definition of Ackley's function [12] (wikipedia.org) is

$$\min f(x_i, x_{i+1}) = \sum -20 \exp\left[\left] - 0.2\sqrt{0.5(x_i^2 + x_{i+1}^2)}\right]\right. - \exp(0.5(\cos(2\pi x_i) + \cos(2\pi x_{i+1})) + e + 20$$

where $i = 1, 3, 5, \ldots, n-1$. n is the number of variable. It is a single objective, fundamentally 2 variable system. The variable range used is $-5 \le x_i \le 5$. The function minimum is 0 at $x_i = 0$. First 2 variables are considered for this function. The settings of VSSCGA for this example are given in Table 7. The results of every digit cycle are shown in Table 8.

Table 7: VSSCGA Settings for Ackley's function with 2 variables

nV	nO	St_{dgt}	Sp_{dgt}	nP	Max_{Itr}	Max_{Rnd}	Pc	Pm
2	1	1	4	20	50	4	0.9	0.1

Table 8: Digit Cycle Results for Ackley's function with 2 variables

nD	Solution x_i	Function Value f_i	Fitness Value	(Digit Cycle, Round, Iteration) at which the solution obtained
1	0, 0	0	1	1, 1, 1
2	0, 0	0	1	
3	0, 0	0	1	
4	0, 0	0	1	

Second 4 variables are considered for this function. The settings of VSSCGA for this example are given in Table 9. The results of every digit cycle are shown in Table 10.

Table 9: VSSCGA Settings for Ackley's function with 4 variables

nV	nO	St_{dgt}	Sp_{dgt}	nP	Max_{Itr}	Max_{Rnd}	Pc	Pm
4	1	1	4	20	50	4	0.9	0.1

Table 10: Digit Cycle Results for Ackley's function with 4 variables

nD	Solution x_i	Function Value f_i	Fitness Value	(Digit Cycle, Round, Iteration) at which the solution obtained
1	0, 0,0,0	0	1	1, 1, 27
2	0, 0,0,0	0	1	
3	0, 0,0,0	0	1	
4	0, 0,0,0	0	1	

At last 8 variables are considered for this function. The settings of VSSCGA for this example are given in Table 11. The results of every digit cycle are shown in Table 12.

Table 11: VSSCGA Settings for Ackley's function with 4 variables

nV	nO	St_{dgt}	Sp_{dgt}	nP	Max_{Itr}	Max_{Rnd}	Pc	Pm
8	1	1	4	20	50	4	0.9	0.1

Table 12: Digit Cycle Results for Ackley's function with 4 variables

nD	Solution x_i	Function Value f_i	Fitness Value	(Digit Cycle, Round, Iteration) at which the solution obtained
1	(0, 0, 0, 0, 0, 0, 0, 0)	0	1	1, 4, 43
2	(0, 0, 0, 0, 0, 0, 0, 0)	0	1	
3	(0, 0, 0, 0, 0, 0, 0, 0)	0	1	
4	(0, 0, 0, 0, 0, 0, 0, 0)	0	1	

4.2.2 Rosenbrock's function

The function definition of Rosenbrock's function (wikipedia.org) is

$$\min f(x_i, x_{i+1}) = \sum \left[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2\right]$$

where $i = 1, 3, 5, \ldots, n-1$ It is a single objective, fundamentally 2 variable system. The variable range used is$-5 \le x_i \le 5$. The function minimum is 0 at $x_i = 0$.

First 2 variables are considered for this function. The settings of VSSCGA for this example are given in Table 13 11. The results of every digit cycle are shown in Table 14.

Table 13: VSSCGA Settings for Rosenbrock's function with 2 variables

nV	nO	St_{dgt}	Sp_{dgt}	nP	Max_{Itr}	Max_{Rnd}	Pc	Pm
2	1	1	4	20	50	4	0.9	0.1

Table 14: Digit Cycle Results for Rosenbrock's function with 2 variables

nD	Solution x_i	Function Value f_i	Fitness Value	(Digit Cycle, Round, Iteration) at which the solution obtained
1	(1, 1)	0	1	1, 1, 3
2	(1, 1)	0	1	
3	(1, 1)	0	1	
4	(1, 1)	0	1	

Second **four** variables are considered for this function. The settings of VSSCGA for this example are given in Table 15. The results of every digit cycle are shown in Table 16.

Table 15: VSSCGA Settings for Rosenbrock's function with 2 variables

nV	nO	St_{dgt}	Sp_{dgt}	nP	Max_{Itr}	Max_{Rnd}	Pc	Pm
4	1	1	4	20	50	4	0.9	0.1

Finally **eight** variables are considered for this function. The settings of VSSCGA for this example are given in Table 17. The results of every digit cycle are shown in Table 18.

Table 16: Digit Cycle Results for Rosenbrock's function with 4 variables

nD	Solution x_i	Function Value f_i	Fitness Value	(Digit Cycle, Round, Iteration) at which the solution obtained
1	(1, 1, 1, 1)	0	1	1, 2, 5
2	(1, 1, 1, 1)	0	1	
3	(1, 1, 1, 1)	0	1	
4	(1, 1, 1, 1)	0	1	

Table 17: VSSCGA Settings for Rosenbrock's function with 8 variables

nV	nO	St_{dgt}	Sp_{dgt}	nP	Max_{Itr}	Max_{Rnd}	Pc	Pm
8	1	1	4	20	50	4	0.9	0.1

Table 18: Digit Cycle Results for Rosenbrock's function with 8 variables

nD	Solution x_i	Function Value f_i	Fitness Value	(Digit Cycle, Round, Iteration) at which the solution obtained
1	(1, 1, 1, 1, 1, 1, 1, 1)	0	1	1, 3, 44
2	(1, 1, 1, 1, 1, 1, 1, 1)	0	1	
3	(1, 1, 1, 1, 1, 1, 1, 1)	0	1	
4	(1, 1, 1, 1, 1, 1, 1, 1)	0	1	

4.2.3 Himmelblau function

The function definition of Himmelblau's function (K. Rajan) is

$$\min f(x_i, x_{i+1}) = \sum \left[(x_i^2 + x_{i+1} - 11)^2 + (x_i + x_{i+1}^2 - 7)^2\right]$$

where $i = 1, 3, 5, \ldots, n-1$ This is also a single objective, fundamentally 2 variable system. The variable range used is $0 \le x_i \le 10$. The function minimum is 0 at $x_i = (3, 2)$ for $n = 2$.

First 2 variables are considered for this function. The settings of VSSCGA for this example are given in Table 19. The results of every digit cycle are shown in Table 20.

Table 19: VSSCGA Settings for Himmelblau's function with 2 variables

nV	nO	St_{dgt}	Sp_{dgt}	nP	Max_{Itr}	Max_{Rnd}	Pc	Pm
2	1	1	4	20	50	4	0.9	0.1

Table 20: Digit Cycle Results for Himmelblau's function with 2 variables

nD	Solution x_i	Function Value f_i	Fitness Value	(Digit Cycle, Round, Iteration) at which the solution obtained
1	(3, 2)	0	1	1, 1, 3
2	(3, 2)	0	1	
3	(3, 2)	0	1	
4	(3, 2)	0	1	

Second **four** variables are considered for this function. The settings of VSSCGA for this example are given in Table 21. The results of every digit cycle are shown in Table 22.

Table 21: VSSCGA Settings for Himmelblau's function with 4 variables

nV	nO	St_{dgt}	Sp_{dgt}	nP	Max_{Itr}	Max_{Rnd}	Pc	Pm
4	1	1	4	20	50	4	0.9	0.1

Table 22: Digit Cycle Results for Himmelblau's function with 4 variables

nD	Solution x_i	Function Value f_i	Fitness Value	(Digit Cycle, Round, Iteration) at which the solution obtained
1	(3, 2, 3, 2)	0	1	1, 1, 17
2	(3, 2, 3, 2)	0	1	
3	(3, 2, 3, 2)	0	1	
4	(3, 2, 3, 2)	0	1	

Finally **eight** variables are considered for this function. The settings of VSSCGA for this example are given in Table 23. The results of every digit cycle are shown in Table 24.

Table 23: VSSCGA Settings for Himmelblau's function with 8 variables

nV	nO	St_{dgt}	Sp_{dgt}	nP	Max_{Itr}	Max_{Rnd}	Pc	Pm
8	1	1	4	20	50	4	0.9	0.1

Table 24: Digit Cycle Results for Himmelblau's function with 8 variables

nD	Solution x_i	Function Value f_i	Fitness Value	(Digit Cycle, Round, Iteration) at which the solution obtained
1	(3, 2, 3, 2, 3, 2, 3, 2)	0	1	1, 2, 35
2	(3, 2, 3, 2, 3, 2, 3, 2)	0	1	
3	(3, 2, 3, 2, 3, 2, 3, 2)	0	1	
4	(3, 2, 3, 2, 3, 2, 3, 2)	0	1	

4.2.4 Powel's function

The function definition of Powel's function (K. Rajan) is

$$\min f(x_i, x_{i+1}, x_{i+2}, x_{i+3}) = \sum \left[(x_i + 10x_{i+1})^2 + 5(x_{i+2} - x_{i+3})^2\right] + \left[(x_{i+1} - 2x_{i+2})^4 + 10(x_i - x_{i+3})^4\right]$$

where $i = 1, 3, 5, \ldots, n-3$ The minimum of $f(x)$ is 0 at $x = (0, 0, 0, 0)$ for $n = 4$. First 4 variables are considered for this function. The settings of VSSCGA for this example are given in Table 25. The results of every digit cycle are shown in Table 26.

Table 25: VSSCGA Settings for Powel's function with 4 variables

nV	nO	St_{dgt}	Sp_{dgt}	nP	Max_{Itr}	Max_{Rnd}	Pc	Pm
4	1	1	4	50	100	4	0.9	0.1

Table 26: Digit Cycle Results for Powel's function with 4 variables

nD	Solution x_i	Function Value f_i	Fitness Value	(Digit Cycle, Round, Iteration) at which the solution obtained
1	(0, 0, 0, 0)	0	1	1, 1, 24
2	(0, 0, 0, 0)	0	1	
3	(0, 0, 0, 0)	0	1	
4	(0, 0, 0, 0)	0	1	

Finally 8 variables are considered for this function. The settings of VSSCGA for this example are given in Table 27. The results of every digit cycle are shown in Table 28.

Table 27: VSSCGA Settings for Powel's function with 8 variables

nV	nO	St_{dgt}	Sp_{dgt}	nP	Max_{Itr}	Max_{Rnd}	Pc	Pm
8	1	1	4	50	100	4	0.9	0.1

Table 28: Digit Cycle Results for Powel's function with 8 variables

nD	Solution x_i	Function Value f_i	Fitness Value	(Digit Cycle, Round, Iteration) at which the solution obtained
1	(0, 0, 0, 0, 0, 0, 0, 0)	0	1	1, 2, 74
2	(0, 0, 0, 0, 0, 0, 0, 0)	0	1	
3	(0, 0, 0, 0, 0, 0, 0, 0)	0	1	
4	(0, 0, 0, 0, 0, 0, 0, 0)	0	1	

4.2.5 Schaffer 4 function

The Schaffer 4 function (wikipedia.org) [12] is also a good example to test VSSCGA for its ability to get the solution very precisely i.e. it's converging feature to very accurate solution. The function definition is

$$\min f(x_1, x_2) = 0.5 + \frac{\cos^2(\sin(x_1^2 - x_2^2) - 0.5}{(1 + 0.001(x_1^2 + x_2^2))^2}$$

These are two variable single objective functions. As per the literature (wikipedia.org) the variable's range is $-100 \le x_i \le 100$, the function value $f(x) = 0.292579$ at $x = (0, 1.25313)$.

First VSSCGA is tested with range given in the literature i.e. $-100 \le x_i \le 100$, and see that whether it is able to get the values of x and f as given. Then to find the values of x and f more precisely, the range $0 \le x_1, x_2 \le 10$ is used. The settings of VSSCGA for the first case are given in Table 29. The results of every digit cycle are shown in Table 30.

Table 29: VSSCGA Settings for Schaffer 4 function's with 2 variables

nV	nO	St_{dgt}	Sp_{dgt}	nP	Max_{Itr}	Max_{Rnd}	Pc	Pm
2	1	1	8	20	50	4	0.9	0.1

Table 30: Digit Cycle Results for Schaffer 4 function's with 2 variables

nD	Solution x_i	Function Value f_i	Fitness Value	(Digit Cycle, Round, Iteration) at which the solution obtained
1	(20, 0)	0.466666411053995	0.681818300646408	1, 1, 2
2	(6, 0)	0.313126889410540	0.761541027043394	2, 2, 16
3	(1.2, −5)	0.302509651423065	0.767748629660781	3, 1, 43
4	(−0.48, 1.34)	0.292781321527602	0.773526027447829	4, 3, 78
5	(−0.022, 1.254)	0.292580331426205	0.773646307070619	5, 4, 94
6	(−0.0356, 1.254)	0.292580056589184	0.773646471568474	6, 2, 67
7	(−2.0e−005, 1.25312)	0.292578632434833	0.773647323966897	7, 4, 100
8	(−1.99e−006, 1.2531319)	0.292578632036065	0.773647324205572	8, 2, 94

The result shows that the VSSCGA provides promising results. Then it is decided to estimate the value of the function more precisely than the value given in literature (wikipedia.org) (infinity77.net) [13]. The settings of VSSCGA for this case are given in Table 31. The results of every digit cycle are shown in Table 32.

Table 31: VSSCGA Settings for Schaffer 4 function (for precise value finding)

nV	nO	St_{dgt}	Sp_{dgt}	nP	Max_{Itr}	Max_{Rnd}	Pc	Pm
2	1	1	10	50	100	4	0.9	0.1

Table 32: Digit Cycle Results for Schaffer 4 function (for precise value finding)

nD	Solution x_i	Function Value f_i	Fitness Value	(Digit Cycle, Round, Iteration) at which the solution obtained
1	(1, 3)	0.305557784250938	0.765956139255643	1, 1, 1
2	(0.3, 1.3)	0.293051770390249	0.773364240240896	2, 1, 9
3	(0.13, 1.26)	0.292592689956555	0.773638910207368	3, 1, 74
4	(0, 1.253)	0.292578681508000	0.773647294595127	4, 2, 90
5	(0, 1.2531)	0.292578634920471	0.773647322479168	5, 1, 70
6	(0, 1.25313)	0.292578632045530	0.773647324199907	6, 1, 79
7	(0, 1.253132)	0.292578632036061	0.773647324205574	7, 1, 87
8	(0, 1.2531318)	0.292578632035983	0.773647324205621	8, 2, 84
9	(0, 1.25313183)	0.292578632035981	0.773647324205622	9, 1, 75
10	(0, 1.253131834)	0.292578632035980	0.773647324205622	10, 1, 76

Here it is found that the minimum value of Schaffer 4 function is 0.292578632035980 at (0, 1.253131834). The digit cycle setting Sp_{dgt} is set to 10, because the fitness value and function values have saturated at Matlab's capacity.

4.3 Benchmark Problems

Now it is time to verify the capability of VSSCGA with benchmark problems with more dimensions and more number of objectives.

4.3.1 Interval arithmetic benchmark

First the interval arithmetic benchmark problem (Abraham) has been taken to test VSSCGA. The system of equations of interval arithmetic benchmark is,

$$
\begin{aligned}
f_1(x) &= x_1 - 0.18324757x_3x_4x_9 - 0.25428722 = 0\\
f_2(x) &= x_2 - 0.16275449x_1x_{10}x_6 - 0.37842197 = 0\\
f_3(x) &= x_3 - 0.16955071x_1x_{10}x_2 - 0.27162577 = 0\\
f_4(x) &= x_4 - 0.15585316x_6x_7x_1 - 0.19807914 = 0\\
f_5(x) &= x_5 - 0.19950920x_6x_7x_3 - 0.44166728 = 0\\
f_6(x) &= x_6 - 0.18922793x_5x_8x_1 - 0.14654113 = 0\\
f_7(x) &= x_7 - 0.21180486x_5x_8x_2 - 0.42937161 = 0\\
f_8(x) &= x_8 - 0.17081208x_6x_7x_1 - 0.07056438 = 0\\
f_9(x) &= x_9 - 0.19612740x_6x_8x_{10} - 0.34504906 = 0\\
f_{10}(x) &= x_{10} - 0.21466544x_4x_8x_1 - 0.42651102 = 0
\end{aligned}
$$

This system has 10 variables and 10 objective functions. Crina Grosan and Ajith Abraham [3] have reported 8 solutions for this problem. Among those the second solution shows least (minimum) function values. Hence this second solution is compared with the final (i.e. last digit cycles) results of VSSCGA. The variable range considered for this case is $-2 \le x \le 2$, $i = 1, 2, \ldots, 10$.

Table 33: VSSCGA Settings for Interval arithmetic benchmark problem

nV	nO	St_{dgt}	Sp_{dgt}	nP	Max_{Itr}	Max_{Rnd}	Pc	Pm
10	10	6	10	100	100	4	0.9	0.1

The settings of VSSCGA for this case are given in Table 33. The comparison of results of VSSCGA with Crina Grosan and Ajith Abraham's second solution [3] (Abraham) is shown in Table 34.

4.3.2 Neurophysiology Application

The next benchmark problem considered is the Neurophysiology Application (Abraham). This system has 6 variables and 6 objective functions. The system of equations of Neurophysiology Application benchmark is,

$$
\begin{aligned}
f_1(x) &= x_1^2 + x_3^2 - 1 = 0\\
f_2(x) &= x_2^2 + x_4^2 - 1 = 0\\
f_3(x) &= x_5x_3^3 + x_6x_4^3 = 0\\
f_4(x) &= x_5x_1^3 + x_6x_2^3 = 0\\
f_5(x) &= x_5x_1x_3^2 + x_6x_2x_4^2 = 0\\
f_6(x) &= x_5x_3x_1^2 + x_6x_4x_2^2 = 0
\end{aligned}
$$

Table 34: Comparison of VSSCGA's results with Crina Grosan and Ajith Abraham's results for Interval arithmetic benchmark problem

	Crina Grosan and Ajith Abraham's second solution (Abraham)[3]		Results of VSSCGA	
i	Solution x_i	Function Value f_i	Solution x_i	Function Value f_i
1	0.1224819761	0.1318552790	0.25783196	−0.000001093119041
2	0.1826200685	0.1964428361	0.379999992	−0.001096214219632
3	0.2356779803	0.0364987069	0.278724	0.000001618504025
4	−40.0371150470	0.2354890155	0.2006688	0.000000078655073
5	0.3748181856	0.0675753064	0.44399996	−0.001250878418472
6	0.2213311341	0.0739986588	0.149176	0.000000387694704
7	0.0697813035	0.3607038292	0.43199452	−0.000000091118403
8	0.0768058043	0.0059182979	0.073399996	−0.000002515583961
9	−0.0312153867	0.3767487763	0.345959968	−0.000006503849487
10	0.1452667120	0.2811693568	0.4271998388	−0.000126401096904
			Fitness Value	0.997520381531896

Crina Grosan and Ajith Abraham have reported 12 solutions for this problem. Among those the twelfth solution shows least (minimum) function values. Hence this twelfth solution is compared with the final results of VSSCGA. On reviewing the results of Crina Grosan and Ajith Abraham, the variable range kept for this case is $-1 \le x_i \le 1$, $i = 1, 2, \ldots, 10$ and the settings of VSSCGA for this case are given in Table 35. The comparison of results of VSSCGA with Crina Grosan and Ajith Abraham's twelfth solution (Abraham) is shown in Table 36.

Table 35: VSSCGA Settings for Neurophysiology Application (case 2)

nV	nO	St_{dgt}	Sp_{dgt}	nP	Max_{Itr}	Max_{Rnd}	Pc	Pm
6	6	5	10	200	100	5	0.9	0.1

Table 36: Comparison of VSSCGA's results with Crina Grosan and Ajith Abraham's results for Neurophysiology Application problem (case 1)

	Crina Grosan and Ajith Abraham's second solution (Abraham) [3]		Results of VSSCGA	
i	Solution x_i	Function Value f_i	Solution x_i	Function Value f_i
1	−0.807866890	0.0050092197	0.556000066	0.000029529993604
2	−0.9560562726	0.0366973076	−0.647718	−0.000000258076000
3	0.5850998782	0.0124852708	−0.831200014	−0.000004409545462
4	−0.2219439027	0.0276342907	−0.76188	0.000007785476757
5	0.06620152964	0.0168784849	0.0000199	0.000013698338141
6	−0.0057942792	0.0248569233	−0.000016	−0.000000024868871
			Fitness Value	0.999944296804184

Here one more set of results of VSSCGA are given, which shows that as like GA, the VSSCGA also gives better results over time. The settings of VSSCGA for this case are given in Table 3.41 37. The comparison of results of VSSCGA with Crina Grosan and Ajith Abraham's twelfth solution is shown Table 38. All the above test cases show that the VSSCGA is giving very precise and promising results. The choice of the various pa-

Table 37: VSSCGA Settings for Neurophysiology Application (case 2)

nV	nO	St_{dgt}	Sp_{dgt}	nP	Max_{Itr}	Max_{Rnd}	Pc	Pm
6	6	5	10	100	200	5	0.9	0.1

Table 38: Comparison of VSSCGA's results with Crina Grosan and Ajith Abraham's results for Neurophysiology Application problem (case 1)

	Crina Grosan and Ajith Abraham's second solution (Abraham)[3]		Results of VSSCGA	
i	Solution x_i	Function Value f_i	Solution x_i	Function Value f_i
1	−0.8078668904	0.0050092197	−0.73834	0.000000000006542
2	−0.9560562726	0.0366973076	−0.271426396	0.000000000110189
3	0.5850998782	0.0124852708	0.67442868	−0.000000000423724
4	−0.2219439027	0.0276342907	−0.9624592	0.000000000318003
5	0.06620152964	0.0	0.000000000218384	0.000013698338141
6	−0.0057942792	0.0248569233	0.0	−0.000000000308311
			Fitness Value	0.999999998614848

rameters in VSSCGA varies with the problem at hand. However, from our experiments we infer that like all other GA algorithms it is very easy to find the appropriate settings for VSSCGA too. The advantage we get here is that now we have a variable search space and that intuitively will converge faster than that of traditional GAs.

5 Conclusion

A new Genetic Algorithm that can vary its search space size and find the solution of non- linear equations system very precisely is introduced. A new Inverse Ranked Roulette Wheel selection is introduced to overcome the premature convergence. The binary imitative integer coded uniform cross over and mutations are adopted. The adopted mutation decreases the execution time. Elitism and Migration are adopted after some predefined iterations, and thus improves the randomness and carry the best solution throughout the run.

Even though it can solve many problems, some benchmark problems are tried to test the performance of the VSSCGA and very good and more precise results are obtained. VSSCGA confirms that it can handle multi variable, multi objective and high search space problems very efficiently.

In future variable rate mutation may be integrated with this GA to analyze its performance. The VSSCGA may be used for other applications where other optimization techniques do not give good results.

References

[1] Akira Oyama, Shigeru Obayashi and Kazuhiro Nakahashi, "*Real-coded adaptive range genetic algorithm and its application to aero dynamic design*", JSME International Journal series A-solid Mechanics and Material Engineering, Vol 43, 124–129, 2000.

[2] F. Herrera and M. Lozano, *Two-loops real-coded genetic algorithms with adaptive control of mutation step sizes*, Applied intelligence, Vol 13, 187–204, 2000.

[3] Crina Grosan and Ajith Abraham, *A New Approach for Solving Nonlinear Equations Systems*, IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems And Humans, Vol. 38, No. 3, 2008.

[4] Satoshi Tomioka, Shusuke Nisiyama, and Takeaki Enoto, *Non-linear least square regression by adaptive domain method with multiple genetic algorithms*. IEEE Transactions of evolutionary computation, Vol 11, IS 1, 1–16, 2007.

[5] Julia Carson *et al. Genetic Algorithms: Advances in Research and Applications* (Computer Science, Technology and Applications, Nova Science Pub Inc, 2017.
[6] J.H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, Michigan, 1975; re-issued by MIT Press, 1992.
[7] John R Koza, *Genetic Algorithms and Genetic Programming*, Power Point Presentation, Department of Electrical Engineering, School of Engineering, Stanford University, http://www.genetic-programming.com/coursemainpage.html 2003.
[8] Mitchell Melanie, *An Introduction to Genetic Algorithms*, A Bradford Book, The MIT Press, Cambridge, 1999.
[9] Jorge Magalhães-Mendes, *A Comparative Study of Crossover Operators for Genetic Algorithms to Solve the Job Shop Scheduling Problem*, WSEAS Transactions on Computers, Issue 4, Volume 12, 2013.
[10] K.A. De Jong, *An analysis of the behavior of a class of genetic adaptive systems*, Doctoral dissertation, University of Michigan, Ann Arbor, Michigan, 1975.
[11] S. Effati and A.R. Nazemi, *A new method for solving a system of the nonlinear equations*, Applied Mathematics and Computation, vol. 168, no. 2, 877–894, 2005.
[12] http://en.wikipedia.org/wiki/Test_functions_for_optimization
[13] http://infinity77.net/global_optimization/test_functions_nd_S.html
[14] Turgut, O. E., Turgut M. S., Coban M. T., *Chaotic quantum behaved particle swarm optimization algorithm for solving nonlinear system of equations*, Computers and Mathematics with Applications, Vol. 68, 508–530, 2014.
[15] Pourjafari E., Mojallali H., *Solving nonlinear equations systems with a new approach based on invasive weed optimization algorithm and clustering*, Swarm and Evolutionary Computation, Vol. 4, 33–43, 2012.
[16] Hirsch M. J., Pardalos, P. M., Resende M. G. C., *Solving systems of nonlinear equations with continuous GRASP, Nonlinear analysis: Real World Applications*, Vol. 10, 2000–2006, 2009.
[17] Abd-El-Wahed, W.F., Mousa, A.A., El-Shorbagy, M.A.: *Integrating particle swarm optimization with genetic algorithms for solving nonlinear optimization problems*. J. Comput. Appl. Math. 235, 1446–1453, 2011.
[18] Bianchini, M., Fanelli, S.: *Optimal algorithms for well-conditioned nonlinear systems of equations*. IEEE Trans. Comput. 50(7), 689–698, 2001.
[19] Joshi, G., Krishna, M.B.: *Solving system of non-linear equations using genetic algorithm*. In: International Conference on Advances in Computing, Communications and Informatics (ICACCI) , 2014.
[20] Konaka, A., Coitb, D.W., Smith, A.E.: *Multi-objective optimization using genetic algorithms: a tutorial*. Reliab. Eng. Syst. Saf. 91, 992–1007, 2006.
[21] Pourrajabian, A., Ebrahimi, R., Mirzaei, M., Shams, M.: *Applying genetic algorithms for solving nonlinear algebraic equations*. Appl. Math. Comput. 219, 11483–11494, 2013
[22] Raja, M.A.Z., Sabir, Z., Mehmood, N., Al-Aidarous, E.S., Khan, J.A.: *Design of stochastic solvers based on genetic algorithms for solving nonlinear equations*. Neural Comput. Appl. 26, 1–23, 2015.
[23] Ren, H., Wua, L., Bi, W., Argyros, I.K.: *Solving nonlinear equations system via an efficient genetic algorithm with symmetric and harmonious individuals*. Appl. Math. Comput. 219, 10967–10973, 2013.
[24] D Mishra, PK Kalra , *Modified Hopfield Neural Network Approach for Solving Nonlinear Algebraic Equations*. - Engineering Letters, 2007
[25] Deepak Mishra and Prem K. Kalra, *An energy function approach for finding roots of characteristic equation*, ictact journal on soft computing: special issue on fuzzy in industrial and process automation, volume: 02, issue: 01 237, 2011.
[26] Chhavi Mangla Harsh Bhasin Musheer Ahmad Moin Uddin, *Novel Solution of Nonlinear Equations Using Genetic Algorithm*, Industrial Mathematics and Complex Systems, pp 249-257, 2017.
[27] Nyoman Gunantara, A review of multi-objective optimization: Methods and its applications, Cogent Engineering, 5: 1502242, 2018

Appendix A: The proposed algorithm of the VSSCGA

The algorithm of the VSSCGA is.

1. Initialize GA parameters,
   (a) Number of Population (nP),
   (b) Number of Variables (nV),
   (c) Number of Objectives (nO),
   (d) Start Digit (St_{Dgt}),
   (e) Stop Digit (Sp_{Dgt}),
   (f) Maximum Iterations (Max_{Itr}),

(g) Maximum Rounds (Max_{Rnd}),
(h) Cross Over Probability (Pc),
(i) Mutation Probability (Pm),
(j) Variables range minimum ($Vmin$) and maximum ($Vmax$), Number of Population Individuals (candidates) to be preserved for Elitism ($nElit$),
(k) Number of candidates to be carried forward for Migration ($nMigr$),
(l) Elitism starting Iteration ($Elit_{Itr}$),
(m) Migration injection Iteration ($Migr_{Itr}$) and
(n) Minimum Fitness (min_{Fit}) value to be reached before the stopping criterions effective, except the Max_{Itr}.
(o) Number of Iterations ($nCons_{Itr}$) to be verified for the Best Fit consistency among Iterations, as stopping criterion of Iterations.
(p) Number of Population Individuals ($nSatu_{Pop}$) to be verified for the Fitness Saturation among Populations, as stopping criterion of Iterations.
(q) Number of Rounds ($nCons_{Rnd}$) to be verified for the Best Fit consistency among Rounds, as stopping criterion of Rounds.

2. Assign 'Number of Digit (nD) = St_{Dgt}'.
3. While '$nD \le Sp_{Dgt}$', carry out the following steps 4 to 25, else produce the final Results. [Outer Loop].
4. Assign '$Round(Rnd) = 1$'.
5. While '$Rnd \le Max_{Rnd}$', carry out the following steps 6 to 23, else produce the current Digit Cycle's Results. [Middle Loop].
6. Randomly initialize new Population (Pop) matrix with dimension $nV \times nD \times nP$, with numbers ranging from 0 to 9.
7. Assign '$Itr = 1$'.
8. While stopping criterions not met carry out the following steps 9 to 20, else produce the current Round's Results. [Inner Loop].
9. If '$[(nD = St_{Dgt} \&\& Rnd > 1) || (nD \neq St_{Dgt})] \, \& \, \& \, (Itr = Migr_{Itr})$', do Migration.
10. Compute 'Decimal Value (Dv)' and 'Value in Range (Rv)' for each Variable of the Individual of the total Population.
11. Compute 'Objective Values (Obj)' and 'Fitness Value (Fit)' for each Population Individual (Pop_{Idvl}).
12. If '$Itr > Elit_{Itr}$', do Elitism.
13. Find Maximum of the Fitness Value (max_{Fit}) and it's corresponding Population Individual (max_{Pop}).
14. If '$Itr = 1$', assign '$Best_{Fit} = max_{Fit}$ and $Best_{Pop} = max_{Pop}$'. Else, if '$Best_{Fit} < max_{Fit}$', assign '$Best_{Fit} = max_{Fit}$ and $Best_{Pop} = max_{Pop}$'.
15. If '$Itr \ge MaxItr$', Stop the Inner Loop. Go to Step 21.
16. If '$Best_{F}it > min_{F}it$', and $Itr > Elit_{Itr}$ & $Itr > Migr_{Itr}$
17. And if $Best_{Fit}$ is consistent over $nCons_{Itr}$, Stop the Inner Loop.
Go to Step 21.
And if Fitness value of each Pop_{Idvl} is saturated over $nSatu_{Pop}$, Stop the Inner Loop. Go to Step 21.
18. Select Pop_{Idvl} from the total Population by Inverse Ranked Roulette Wheel (IRRW).
19. Generate Mating pool and do Binary Imitative Single Point Uniform Cross Over with probability Pc.
20. Do Binary Imitative Single Point Uniform Mutation with probability Pm.
21. Assign '$Itr = Itr + 1$', go to step 8.
22. If '$Run = 1$', assign '$DBest_{Fit} = Best_{Fit}$ and $DBest_{Pop} = Best_{Pop}$'. Else, if '$DBest_{Fit} < Best_{Fit}$', assign '$DBest_{Fit} = Best_{Fit}$ and $DBest_{Pop} = Best_{Pop}$'.
23. If $DBest_{Fit}$ is consistent over $nCons_{Rnd}$, Stop the Middle Loop. Go to Step 24.
24. Assign '$Rnd = Rnd + 1$', go to step 5.
25. If '$nD = St_{Dgt}$', assign '$FBest_{Fit} = DBest_{Fit}$ and $FBest_{Pop} = DBest_{Pop}$'. Else, if '$FBest_{Fit} < DBest_{Fit}$', assign '$FBest_{Fit} = DBest_{Fit}$ and $FBest_{Pop} = DBest_{Pop}$'.
26. Assign '$nD = nD + 1$', go to step 3.